MAINTAINING A SPARSE INVERSE IN THE SIMPLEX METHOD

BY

JOHN A. TOMLIN

TECHNICAL REPORT NO. 70-16
NOVEMBER 1970

OPERATIONS
RESEARCH
HOUSE

Stanford
University
CALIFORNIA

35

MAINTAINING A SPARSE INVERSE IN THE SIMPLEX METHOD

by

John A. Tomlin

Technical Report No. 70-16
November 1970

DEPARTMENT OF OPERATIONS RESEARCH
Stanford University
Stanford, California

Research and reproduction of this report was partially supported by Office of Naval Research, Contract N-00014-67-A0112-0011; U.S. Atomic Energy Commission, Contract AT[04-3]326 PA #18; National Science Foundation Grant GP 9329.

1. INTRODUCTION

Sparse matrix methods have been important in practical linear programming from the very earliest implementations of the revised simplex method. These methods have become even more important as the size of problems presented for solution has continued to grow until, at present, linear programs with a few thousand constraints are quite commonplace. Fortunately, and not surprisingly, the densities of the constraint matrices decline with the increase in dimension thus keeping the amount of data storage and arithmetic within possible proportions.

Techniques for handling sparse matrices are important in two major areas in linear programming:-

1. Storage and retrieval of the problem data. There are several techniques for storing sparse matrices in use at present (see Smith [13] and de Buchet [6]).

2. Maintaining and applying the basis inverse or substitute inverse.

This paper is concerned with the second area. Readers will be assumed to be familiar with the simplex method using the product form of inverse (see Dantzig [7]) and the Gaussian elimination for solving systems of equations (see Forsythe and Moler [10]).

In recent years it has been realized that the sparsity of the inverse representation produced by the linear programming inversion routine can be drastically improved by using the Gaussian or Elimination form of inverse (E.F.I.) rather than using the Gauss-Jordan or product form of inverse (P.F.I.). This scheme was first advocated by Markowitz [11] and subsequently for the special case of staircase matrices by Dantzig [8]. Later still Dantzig, Harvey, McKnight and Smith [9] experimentally demonstrated the superiority of the E.F.I. over the P.F.I. for a number of linear programming problems and Brayton, Gustavson and Willoughby [5] were able to prove this for general sparse matrices.

The elimination form of inverse is now known to be implemented in at least two commercial linear programming codes; Standard Oil's M5 code [9] and Scientific Control Systems' UMPIRE code (see Beale [3]). Other codes are in the course of being modified and may have been already. Some aspects of the P.F.I. and E.F.I. inversion techniques will be examined in the next section.

The success of the Gaussian or Elimination form of inverse naturally leads one to ask whether the resulting triangular factors can be used to advantage in updating the inverse in the iterations following a re-inversion. The answer would certainly appear to be yes. Dantzig [8] advocated updating the triangular factors of the basis for his staircase algorithm since this preserves the staircase form of the factors. It seems extremely likely that in this special case updating the factors rather than using the ordinary product form of updating would lead to an increase in efficiency. Bennett and Green [4] pursued this technique for general sparse matrices and Bartels [1] and Bartels and Golub [2] have advocated a form of Dantzig's method on the grounds of numerical stability, without however commenting on the sparsity implications. Brayton et al [5] have proposed two other updating techniques although their concern is not primarily with efficiency in terms of the simplex method.

The three new proposals will be reviewed and compared in Section 3.together with their implications for the simplex method. One of these methods is then selected for some computational experiments, the results of which are presented in Section 4. Section 5 describes how this method might be incorporated efficiently into a production code.

2. THE PRODUCT AND ELIMINATION FORMS OF INVERSE

P.F.I.

A popular method of carrying out re-inversion (see Orchard-Hays [12]) is to order the rows (implicitly) and the columns (explicitly) of B and partition as follows:-

$$B = \begin{bmatrix} \Lambda_1 & & \\ A & \tilde{B} & \\ D & E & \Lambda_2 \end{bmatrix}$$

where Λ_1 and Λ_2 are lower triangular. The submatrix $\tilde{B}$ is sometimes referred to as the "bump" and those vectors with elements in Λ_1 are "above the bump", those in Λ_2 "below the bump". It seems to be characteristic of most linear programs that the submatrices Λ_1 and particularly Λ_2 are quite substantial portions of B.

The above partitioned form of B may be factorized thus

$$B = \begin{bmatrix} \Lambda_1 & & \\ A & I & \\ D & & I \end{bmatrix} \begin{bmatrix} I & & \\ & B & \\ & & I \end{bmatrix} \begin{bmatrix} I & & \\ & I & \\ & E & I \end{bmatrix} \begin{bmatrix} I & & \\ & I & \\ & & \Lambda_2 \end{bmatrix}$$

Since Λ_1 and Λ_2 are triangular the calculation of the product form of inverse is non-trivial only for the submatrix $\tilde{B}$. Various "merit" schemes have been proposed for selecting the order of pivots in B but they will not be discussed here (see e.g. Tewarson [15]).

Using the notation of reference 5 we may then express B as a product of elementary transformations

$$B = T_1 T_2 \dots T_\ell$$

or equivalently

$$B^{-1} = T_\ell^{-1} \dots T_2^{-1} T_1^{-1}$$

where T_k is of the form (we assume B is m x m)

$$T_k = \begin{bmatrix} 1 & & & t_{1k} & & & \\ & \ddots & & \vdots & & & \\ & & 1 & \vdots & & & \\ & & & \ddots \vdots & & & \\ & & & t_{kk} & & & \\ & & & \vdots & \ddots & & \\ & & & \vdots & & 1 & \\ & & & \vdots & & & \ddots \\ & & & t_{mk} & & & 1 \end{bmatrix}$$

$$= I + (t_k - e_k) e_k'$$

where e_k is the kth unit m vector, e_k' its transpose, and

$$t_k = T_{k-1}^{-1} \dots T_2^{-1} T_1^{-1} b_k$$

where b_k is the kth column of B.

Note however that two transformations are made for each column in the "bump", hence the elements of the T_k for such columns are zero on the rows corresponding to Λ_2 and in addition a second transformation, with a unit pivot and the kth column of E on the rows of Λ_2, is also made. Note also that $T_k^{-1} = I - t_{kk}^{-1} (t_k - e_k) e_k'$ and that it is essentially necessary to keep only the vector t_k to store T_k, and t_k and t_{kk}^{-1} to store T_k^{-1}. (see [5]).

E.F.I.

The above partitioning scheme may be used with only a slight modification in the elimination form of inverse. By rearranging the rows and columns of the submatrices Λ_2, E and D we may repartition B to give (see Beale [3])

$$B = \begin{bmatrix} \Lambda_1 & & \\ \tilde{D} & \tilde{\Lambda}_2 & \tilde{E} \\ A & & \tilde{B} \end{bmatrix}$$

where $\tilde{\Lambda}_2$ is now <u>upper</u> triangular. This procedure obviously facilitates factorization into triangular factors by Gaussian elimination since again only the factorization of $\tilde{B}$ is non-trivial.

Suppose B is now factorized into LU where L is lower and U is upper triangular. Let ℓ_k and u_k be the kth columns of L and U, then defining

$$L_k = I + (\ell_k - e_k)\, e'_k$$
$$U_k = I + (u_k - e_k)\, e'_k$$

we may factorize

$$L = L_1\, L_2 \ldots L_m$$
$$U = U_m\, U_{m-1} \ldots U_1$$

Hence $$B^{-1} = U_1^{-1} \ldots U_m^{-1}\, L_m^{-1} \ldots L_1^{-1}$$

There is some latitude in the choice of values of diagonal elements of L and U. The most efficient choice would appear to be setting u_{kk} to 1 for k in partitions Λ_1, $\tilde{B}$ and ℓ_{kk} to 1 in partition Λ_2. If this is done the number of transformations (disregarding unit transformations) is exactly the same as for the P.F.I. since two transformations are calculated only for the columns in $\tilde{B}$.

The order of choice of pivot elements in $\tilde{B}$ may be decided here, as for the P.F.I., by merit schemes. These are exhaustively discussed by Dantzig et al [9]..

3. METHODS OF UPDATING THE INVERSE

In this section we briefly review, without rigorous proof, and compare four methods of updating the inverse. All except the first assume the elimination form of inverse discussed in the previous section. With the exception of method II all are described at greater length by Brayton et al [5].

Method I [7]

This is the standard procedure used in the product form of the simplex method.

Suppose column b_k of the basis is to be replaced by $\bar{b}_k$. Let B be the original basis (factorized in either E.F.I. or P.F.I. form) and let $\bar{B}$ be the new basis.

Let $$t_{\ell+1} = B^{-1} \bar{b}_k$$

then $$\bar{B}^{-1} = T_{\ell+1}^{-1} B^{-1}$$

where $$T_{\ell+1} = I + (t_{\ell+1} - e_k) e_k'$$

Hence if $$B^{-1} = T_{\ell}^{-1} \dots T_2^{-1} T_1^{-1}$$

then $$\bar{B}^{-1} = T_{\ell+1}^{-1} T_{\ell}^{-1} \dots T_1^{-1}$$

Further changes are made by repeating the above process.

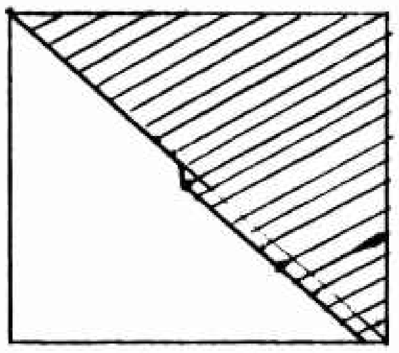
Method II [1]

This is Bartels's version of Dantzig's algorithm [8]. Suppose we have

$$B = LU$$

i.e. $$B^{-1} = U^{-1} L_m^{-1} \dots L_1^{-1}$$

and we again change b_k to $\bar{b}_k$. Then $L^{-1}\bar{B}$ will be identical to U except in column k. Permute columns k+1,...,m one place to the left and place column k in position m to give a Hessenberg matrix

$H =$  $= L^{-1}\bar{B}Q$

(where Q is a permutation matrix).

The elements below the diagonal in columns k to m-1 of H must now be eliminated by a set of elementary transformations. An important feature of Bartels' method is that in carrying out the elimination rows may be interchanged or permuted to place the maximum of the elements $h_{\ell\ell}$, $h_{\ell+1,\ell}$ on the diagonal (Wilkinson's "partial pivoting" strategy [16]) to ensure numerical stability.

The result is a new upper triangular matrix

$$\bar{U} = G_{m-1} P_{m-1} \cdots G_{k+1} P_{k+1} G_k P_k H$$

where the P_ℓ are either unit matrices or the permutation matrices exchanging rows ℓ and $\ell+1$, and G_ℓ are the elementary transformation matrices eliminating the element below the diagonal in column ℓ .

Hence $$\bar{B}^{-1} = Q\,\bar{U}^{-1} G_{m-1} P_{m-1} \cdots G_k P_k L^{-1}$$

Dantzig [8] and Bennett and Green [4] have shown that the "near commutativity" of the matrices G_ℓ may be used to calculate a new lower triangular matrix $\bar{L}$ such that

$$\bar{L}^{-1} = G_{m-1} P_{m-1} \dots G_k P_k L^{-1}$$

Thus maintaining a true triangular decomposition. This is not however an essential feature of the method.

Of course the permutation matrices P_ℓ, Q need not appear explicitly. All that is necessary is to record the order of pivoting. The process is repeated for the next iteration using the new $\bar{U}$ and (implicitly) the new $\bar{L}$.

Method III [5]

Neither this method nor the next maintain a true triangular decomposition. however, both require initially the elimination form of inverse.

Suppose we have B = LU or equivalently

$$B^{-1} = U_1^{-1} \dots U_m^{-1} L_m^{-1} \dots L_1^{-1}$$

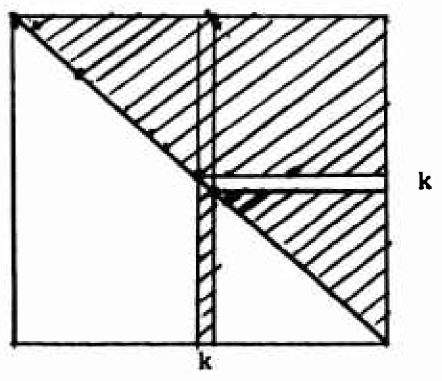
and again change column b_k to $\bar{b}_k$ and B to $\bar{B}$.

The method proceeds as follows:-

Let $$v_k = L^{-1} \bar{b}_k$$

then $L^{-1} \bar{B}$ is identical to U except for column k which is now v_k instead of u_k.

To reduce $L^{-1}\bar{B}$ back to upper triangular form the first step carried out is to reduce row k to zero in columns k+1,...,m.



To accomplish this we form a row transformation using the row vector

$$w_k' = (0,\dots,0,w_{k,k+1},\dots,w_{km}) = (0,\dots,0,u_{k,k+1},\dots,u_{km})\,U^{-1}$$

The row transformation is then the elementary matrix

$$W_k = I + e_k\, w_k'$$

and the desired elimination is performed by forming $W_k^{-1} L^{-1}\bar{B}$, since the columns k+1,...,m of $L^{-1}\bar{B}$ are identical to U, $W_k^{-1}\, U = (I - e_k\, w_k')\, U$, and $w_k'\, U$ is simply the row $(0,\dots,0,u_{k,k+1},\dots,u_{km})$. Furthermore the first k columns of W_k are unit vectors and hence the first k columns of $L^{-1}\bar{B}$ are unchanged.

Now let $t_k = W_k^{-1} L^{-1}\bar{b}_k$ and form the elementary column transformation

$$T_k = I + (t_k - e_k)\, e_k'$$

then clearly forming $T_k^{-1} W_k^{-1} L^{-1}\bar{B}$ reduces column k to the unit vector e_k and columns k+1,...,m are unchanged since they have a zero on the pivot row. The result then is a new upper triangular matrix, say $U^{(k)}$, obtained from U by replacing the kth row and column by the unit vector e_k.

i.e. $$U^{(k)} = T_k^{-1} W_k^{-1} L^{-1} \bar{B}$$

or $$\bar{B}^{-1} = U^{(k)-1} T_k^{-1} W_k^{-1} L^{-1}$$

Using product form notation, if we let $U_\ell^{(k)}$ denote the elementary matrix U_ℓ with $u_{k\ell}$ set to zero we may re-write $\bar{B}^{-1}$ as

$$\bar{B}^{-1} = U_1^{-1} \ldots U_{k-1}^{-1} U_{k+1}^{(k)-1} \ldots U_m^{(k)-1} T_k^{-1} W_k^{-1} L_m^{-1} \ldots L_1^{-1}$$

Note that the row transformation W_k can be represented (if desired) by a product of two-element elementary column transformations. Also note that although we have introduced two new transformations that if U was previously in product form, $U = U_m \ldots U_1$, the new form $U^{(k)}$ is obtained by simply deleting U_k and a single element at most from each $U_{k+1} \ldots U_m$ for a net gain of one transformation in general (unless U_k happens to be the unit matrix).

To carry the process on one simply treats $T_k^{-1} W_k^{-1} L^{-1}$ as the lower triangular factor as before, although in fact if multiplied out the resulting matrix would not in general be triangular.

Method IV [5]

This method, which also assumes the elimination form of inverse, proceeds by inserting matrices T_k into the list $U_m \ldots U_1$ and sometimes deleting a U_k.

To change b_k to $\bar{b}_k$ we form the vector

$$t_k = U_{k+1}^{-1} \ldots U_m^{-1} L_m^{-1} \ldots L_1^{-1} \bar{b}_k$$

and replace U_k by the transformation

$$T_k = I + (t_k - e_k) e_k'$$

Then

$$\bar{B}^{-1} = U_1^{-1} \ldots U_{k-1}^{-1} T_k^{-1} U_{k+1}^{-1} \ldots U_m^{-1} L_m^{-1} \ldots L_1^{-1}$$

Continuation of this process is rather more complicated than in the preceding methods. Suppose now we have changed b_k to $\bar{b}_k$ and wish to change b_ℓ to $\bar{b}_\ell$. There are two cases:-

1. If $\ell < k$ then let

$$t = U_{\ell+1}^{-1} \ldots U_{k-1}^{-1} T_k^{-1} U_{k+1}^{-1} \ldots U_m^{-1} L_m^{-1} \ldots L_1^{-1} \bar{b}$$

form $T_\ell = I + (t_\ell - e_\ell) e_\ell'$

and replace U_ℓ by T_ℓ as before.

2. If $\ell > k$ form

$$t_\ell = T_k^{-1} U_{k+1}^{-1} \ldots U_m^{-1} L_m^{-1} \ldots L_1^{-1} \bar{b}$$

and the transformation T_ℓ as above. However, U_ℓ is not replaced and T_ℓ^{-1} is placed directly before T_k^{-1} in the inverse transformation list.

i.e. $\bar{\bar{B}}^{-1} = U_1^{-1} \ldots U_{k+1}^{-1} T_\ell^{-1} T_k^{-1} U_{k+1}^{-1} \ldots U^{-1} \ldots U_1^{-1} L_m^{-1} \ldots L_1^{-1}$

Continuing further, if columns $k_1, \ldots, k_p$ have been changed and column ℓ is to be altered, let $k = \min (k_1 \ldots k_p)$ and carry out the above procedure.

Having completed our summary of methods for updating a basis inverse or substitute inverse we must consider their relative merits in the context of the simplex method in terms of preserving sparsity, work required per iteration, and storage and data handling considerations.

The great advantage of Method I of course is its simplicity and the minimum of computational steps and housekeeping involved. The essential step in changing the basis in the simplex method, given the constraints

$$Ax = b$$

and the new column a_j to enter the basis, is the calculation of the updated column

$$\alpha = B^{-1} a_j$$

and the ratio test with the updated right-hand side,

$$\beta = B^{-1} b,$$

to find the pivot row k from (assuming $\beta \geq 0$)

$$\theta = \min_{\substack{i \\ \alpha_i > 0}} \frac{\beta_i}{\alpha_i}$$

The vector α, which we cannot avoid calculating, is precisely the vector $t_{\ell+1}$ used in Method I to update the basis inverse (and the right-hand side).

Methods II to IV all suffer from the disadvantage that they require not only the vector α for determining the pivot row but also another vector t_k which is only a partially updated form of a_j. Methods II and III can more or less overcome this disadvantage by putting the vector $L^{-1} a_j$ in temporary storage in the course of calculating α, at the expense of either using more of the computer's core storage (thus effectively limiting the number of vectors which can be saved for multiple pricing) or of using backing store. This burden is, however, comparatively light compared to the immediate and acute difficulty encountered in Method IV where the vector t_k to be calculated for updating purposes cannot in general be known until after the ratio test operation to determine the pivot row. This implies that t_k must therefore be calculated from scratch or obtained by backward transformations on the vector α.

Now let us consider the implications of Methods II to IV in terms of work per iteration expended in updating the basis inverse and of the necessary data manipulation. Current packing methods of storing sparse data, particularly strings of transformations (see [6], [13]) make it rather difficult to modify an existing product form of L and U. Replacement of U_k by a T_k with more non-zeros in Method IV, for example, necessitates either a wholesale shifting of elements of U, maintenance of a second transformation file to be inter-woven with the first, copying out a complete new basis file or initial and wasteful leaving of space for such eventualities. A similar, in fact worse, situation occurs in Method II where on the average half of the U_ℓ transformations must be modified at each iteration, increasing the number of non-zeros to be stored. Explicit storage of U can only be an answer for very small problems. Some kind of two-file system, each taking the new inverse in turn, would appear to be necessary in general, involving a great deal of read/write activity. Method III on the other hand requires only the elimination of previously non-zero elements in U - a comparatively easier task. This point will be returned to in Section 5.

The amount of arithmetic work per iteration would appear to be greatest in Methods II and III. The bulk of the work in Method III comes from applying the G_ℓ transformations to H to reduce it to triangular form. On the average we expect to have to deal with m/2 columns on the right-hand side of the matrix. This is a considerable amount of computation and becomes progressively worse as U fills in with each iteration. For Method III the main effort is in computing the row vector w_k'. It will be shown in Section 5 that this calculation may be performed concurrently with the backward transformation phase of the next simplex iteration. The most important point, however, is that U becomes progressively less and less dense in this method and the calculation progressively easier. There is admittedly the extra complication in Method III that row transformations are called for, but this is more of an inconveneience than a problem and as already pointed out could be circumvented.

Finally we consider the most important point - preservation of sparsity. In their discussion Brayton et al [5] concluded that Method III was superior to Method IV in this respect since both a row and column of U are eliminated in exchange for a vector w_k', which only has elements in positions $k+1,\ldots,m$, and a transformation vector t_k which is the incoming column multiplied only by L^{-1}. On the other hand if in Method IV some columns with very small pivot row index k is introduced then virtually every change thereafter will be of type 2 and we cannot expect much improvement over the ordinary product form method.

In comparing Methods II and III it again seems almost certain that Method III will win out. The number of new elements added in the new transformations would appear to be much the same, but as already mentioned in Method II U becomes more and more dense while in Method III the density of U declines. In this connection it should be pointed out that the columns of U most frequently operated upon are the right-most. These, however, are just those columns which will be initially the most dense after inversion, since inversion schemes generally postpone the densest columns until last.

In the light of these considerations it appears that Method III is the most promising for incorporation into the simplex method, both on the grounds of work per updating and preservation of sparsity. However, there is certainly some increase in updating work per iteration over the standard product form (Method I) and a significant improvement in growth of non-zeros in the basis representation is required to justify its use. Some experimental results are presented in the next section.

Although we have chosen Method III as the best for our purposes this is not to say that the others may not be superior in other circumstances. Bartels's method (II) is of considerable importance in achieving highly accurate solutions to small dense problems (see Bartels and Golub [2]) while Dantzig's original version [8] of Method II for specially structured matrices is as yet untried.

4. COMPUTATIONAL RESULTS

Some computational experiments have been carried out to determine whether Method III gives a sufficient improvement over Method I in terms of growth of non-zeros to warrant further investigation.

To carry out these experiments two linear programming codes were written and run on the Stanford University IBM 360/67. Both are all-in-core FORTRAN codes using the elimination form of inverse. Neither employs multiple pricing. The first code updates the inverse in standard product form fashion (Method I), the second is a modified version using Method III to update the inverse. No attempt has been made to evaluate timings, dependent as they are on machine used and programming technique. Only the number of non-zero elements in the inverse representation is considered.

Five problems ranging from small to moderate have been used. These problems are two of the well-known SHARE standard test problems, 2B and 1B, (see Smith and Orchard-Hays [14]) and 3 other test problems supplied by Joel Cord of IBM. The relevant statistics on these problems are given in Table 1.

Only the smallest problem was run to optimality, the remainder being cut off after 200, 250, 300 and 350 iterations. To compare the two methods the set of problems was run on both codes with a fixed inversion frequency - 40 iterations for the SHARE problems and 50 for the others. The initial comparison is arrived at by counting, at intervals of 10 iterations, the number of new non-zeros added to the inverse since the last re-inversion. To further remove constant factors only non-pivot elements are counted. (Both methods give a net addition of one pivot element per iteration in general.)

All problems were started from a slack basis and hence the iterations up to the first re-inversion are ignored. The number of new non-pivot non-zero elements after multiples of 10 iterations from re-inversion is then averaged for the remainder of the run. These figures are given in Table 2.

The raw data appears in more digestible form in Table 3, where Method I has been used as a base and the fraction of new elements produced by Method III is given. The improvement is obviously very satisfactory, with the possible exception of problem 2. As might be expected the improvement tails off as the number of iterations increases, particularly for the smaller, denser problems.

To obtain a better estimate of the effect of Method III on the simplex algorithm we should also consider the total number of non-pivot non-zeros in the inverse. To this end we list the inversion statistics in Table 4 and compute the fraction of total non-pivot non-zeros produced by Method III, again using Method I as a base. The inversion statistics do not include the original all-slack basis and are of some interest in themselves. The pivot choosing procedure in decomposing $\tilde{B}$ (see Section 2) is essentially algorithm 7 in Dantzig et al [9] .

The modified data for Method III appear in Table 5.

Although the problems are comparatively small and the sample is also small the results in Table 5 are quite encouraging. It appears that for reasonably sparse problems with 200-300 constraints a reduction of 20-30% in the number of non-zeros in the substitute inverse may be expected using Method III, thus considerably reducing the time spent in forward and backward transformation in the simplex method. The results for problem 4 show that the reduction may be dramatic.

5. IMPLEMENTATION OF METHOD III

The results of the preceding section suggest that further experiments using a modified production or commercial code would be worthwhile. It would also appear that the modifications could be made and the extra updating work per iteration accomplished at comparatively little cost.

The first essential is clearly to have a code which employs the elimination form of inverse. Given such a code it should be a simple matter to adjust the storage and buffering activities such that the $T_k^{-1} W_k^{-1}$ transforms can be adjoined to the left of the product form $L_m^{-1} \dots L_1^{-1}$ of L^{-1}.

The reduction of the kth row and column of U to the unit vector e_k can probably be accomplished satisfactorily by simply setting indicator bits. Usually we will have U^{-1} represented as $U_1^{-1} U_2^{-1} \dots U_m^{-1}$, or rather the packed vectors u_ℓ corresponding to these elementary matrices. The reduction may then be accomplished by tagging U_k^{-1} with a bit to indicate it is to be skipped over. Furthermore using, say, the blocked index scheme (see Smith [13]) for storing the vectors u_ℓ the row index k, if it appears, may be tagged to indicate the element is zero for $\ell = k+1,\dots,m$. Failing this the $u_{k\ell}$ elements could be physically replaced by zeros if necessary.

This scheme is of course inefficient in the sense that in an out-of-core system progressively more and more valueless data must be buffered in and out of core. On the other hand if the system is already compute bound rather than I/O bound this is no restriction, and even if this is not the case the very frequent re-inversions carried out for large problems should prevent this inefficiency from reaching really noticeable proportions.

Finally let us consider the process of computing w'_k. For simplicity let us assume we are carrying out the first iteration after a re-inversion. We then have

$$B^{-1} = U_1^{-1} \dots U_m^{-1} L_m^{-1} \dots L_1^{-1}$$

Now the calculation of

$$w'_k = (0,\dots,0,u_{k,k+1},\dots,u_{km})\, U^{-1}$$

and also the modification of U to $U^{(k)}$ may be carried out in a <u>single pass</u> from left to right through the transformations $U_1^{-1} \dots U_m^{-1}$. To see this notice that in computing say

$$(0,\dots,0,z_{k+1},\dots,z_m)\, U_\ell^{-1} \qquad \text{for} \quad \ell > k$$

only the elements $z_{k+1},\dots,z_\ell$ play any part in the arithmetic since U and U_ℓ are upper triangular. The elements $z_{\ell+1},\dots,z_m$ are unchanged. We therefore arrange the computation of w'_k and the modification of U to $U^{(k)}$ as follows:-

<u>Initial Step</u>

Create a zero row vector and skip transformations $U_1^{-1} \dots U_{k-1}^{-1}$. Tag U_k^{-1}. Let $\ell = k+1$.

<u>General Step</u>

For transformation U_ℓ^{-1}, if $u_{k\ell}$ is non-zero extract it and add it in position ℓ of the row vector, marking the element as now being zero in U_ℓ^{-1}. Post-multiply the row vector by U_ℓ^{-1}. Proceed for $\ell = k+1,\dots,m$.

A very advantageous result of this procedure is that the updating of the inverse can be carried out concurrently with the backward transformation of the following simplex iteration. In calculating the pricing vector for the next iteration we must compute

$$\pi' = c' U^{(k)-1} T_k^{-1} W_k^{-1} L^{-1}$$

for some row vector c'. Now the first k-1 columns of $U^{(k)}$ are identical with those of U. The modification of the remaining columns of U may be carried out in the same pass through the transformation file as that for the calculation of π' and furthermore only one unpacking of the non-zeros from each u_ℓ is necessary. Having obtained w_k' the new transformations T_k^{-1}, W_k^{-1} may be attached to L^{-1} and the remainder of the calculation of π' can proceed. Looking to the future, this procedure is admirably suited to parallel processing facilities.

6. CONCLUSION

Linear programming inversion routines have now reached a very high level of sophistication and efficiency, both in terms of speed and sparseness of the resulting inverse. Although there is doubtless still some room for improvement it seems likely that further improvements in efficiency must be sought in other areas. The attempt to maintain a sparse inverse is one such area and the results of Section 4 give considerable grounds for optimism. What is needed now is full scale experimentation with some of the methods reviewed here on large problems of 1000 and more constraints to confirm and extend these results.

ACKNOWLEDGEMENTS

This research was carried out under an IBM Post-Doctoral Fellowship at Stanford University. The author wishes to thank IBM for their support. He is also greatly indebted to George B. Dantzig, Gene H. Golub and Steven Maier for many helpful discussions, and to Roy Harvey, Al Colville and Joel Cord for supplying data for the computational experiments.

REFERENCES

1. Bartels, R.H., "A Numerical Investigation of the Simplex Method", Computer Science Department, Stanford University, Technical Report No. CS-104, July 31, 1968.

2. Bartels, R.H. & Golub, G.H., "The Simplex Method of Linear Programming Using LU Decomposition", Comm.ACM, 12, pp.266-268 and pp.275-278 (1969).

3. Beale, E.M.L., "Sparseness in Linear Programming", Paper presented to the Oxford Symposium on Sparse Systems of Linear Equations, April, 1970.

4. Bennett, J.M. & Green, D.R., "Updating the Inverse or the Triangular Factors of a Modified Matrix", Basser Computing Department, University of Sydney. Technical Report No.42, April, 1966.

5. Brayton, R.K., Gustavson, F.G. & Willoughby, R.A., "Some Results on Sparse Matrices", RC-2332, IBM Research Centre, Yorktown Heights, N.Y., February 14, 1969.

6. de Buchet, J., "How to Take Into Account the Low Density of Matrices to Design a Mathematical Programming Package - Relevant Effects on Optimization and Inversion Algorithms", Paper presented to the Oxford Symposium on Sparse Systems of Linear Equations, April, 1970.

7. Dantzig, G.B., Linear Programming and Extensions, Princeton University Press, Princeton (1963).

8. Dantzig, G.B., "Compact Basis Triangularization for the Simplex Method", pp.125-132 in Recent Advances in Mathematical Programming (R.L. Graves and P. Wolfe, Eds.), McGraw-Hill, New York (1963).

9. Dantzig, G.B., Harvey, R.P., McKnight, R.D., and Smith, S.S., "Sparse Matrix Techniques in Two Mathematical Programming Codes", pp.85-99 in Sparse Matrix Proceedings (R. Willoughby, Ed.), RA-1, IBM Research Centre, Yorktown Heights, N.Y. (1969).

10. Forsythe, G. and Moler, C.B., Computer Solution of Linear Algebraic Equations, Prentice-Hall, Englewood Cliffs, N.J. (1967).

11. Markowitz, H.M., "The Elimination Form of Inverse and its Application to Linear Programming", Man.Sci. 3, pp.255-269 (1957).

12. Orchard-Hays, W., Advanced Linear Programming Computing Techniques, McGraw-Hill, New York (1968).

13. Smith, D.M., "Data Logistics for Matrix Inversion", pp.127-138 in Sparse Matrix Proceedings (R. Willoughby, Ed.), RA-1, IBM Research Centre, Yorktown Heights, N.Y. (1969).

14. Smith, D.M., and Orchard-Hays, W., "Computational Efficiency in Product Form LP Codes", pp.211-218 in Recent Advances in Mathematical Programming (R.L. Graves and P. Wolfe, Eds.), McGraw-Hill, New York (1963).

15. Tewarson, R.P., "On the Product Form of Inverses of Sparse Matrices", SIAM Rev. 8, pp.336-342 (1966).

16. Wilkinson, J.H., Rounding Errors in Algebraic Processes, Prentice-Hall, Englewood Cliffs, N.J. (1963).

Problem Number	1	2	3	4	5
Name	SHARE 2B	SHARE 1B	BRANDY	BANDM	MAR
Rows	99	118	221	306	325
Columns (Structural)	79	225	249	472	452
% Density (of Structural Columns)	10.25	4.45	3.91	1.59	1.77

Table 1

Problem Statistics

Problem		1		2		3		4		5	
Method		I	III	I	III	I	III	I	III	I	III
Iterations since Invert	10	391	176	298	206	717	332	863	252	304	118
	20	891	492	638	497	1589	901	1866	693	628	354
	30	1112	847	993	830	2405	1545	2779	1238	1147	729
	40	1602	1515	1458	1272	3417	2342	3649	1882	1796	1135
	50	-	-	-	-	4254	3062	4646	2642	2261	1441

Table 2

Growth of New Non-Zeros in Basis

Problem		1	2	3	4	5
Iterations since Invert	10	.45	.69	.46	.34	.39
	20	.55	.78	.57	.37	.56
	30	.76	.84	.64	.45	.63
	40	.95	.87	.69	.53	.63
	50	-	-	.72	.58	.64

Table 3

Fraction of New Non-Zeros Produced by Method III

Problem	1	2	3	4	5
Original Basic Non-Zeros	488	498	735	755	822
Non-Pivot Non-Zero Transformation Elements	417	398	595	608	561
Number of Transformations	76	107	133	126	172

Table 4

Inversion Statistics (Averages)

Problem		1	2	3	4	5
Iterations since Invert	10	.73	.87	.71	.58	.78
	20	.70	.86	.68	.53	.77
	30	.83	.88	.71	.54	.75
	40	.96	.90	.73	.59	.72
	50	-	-	.76	.62	.71

Table 5

Fraction of Total Non-Zeros Produced by Method III

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Dept. of Operations Research Stanford University Stanford, Calif.	2a. REPORT SECURITY CLASSIFICATION
	2b. GROUP

3. REPORT TITLE

Maintaining a Sparse Inverse in the Simplex Method

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

TECHNICAL REPORT

5. AUTHOR(S) (Last name, first name, initial)

Tomlin, John A.

6. REPORT DATE November 1970	7a. TOTAL NO. OF PAGES 31	7b. NO. OF REFS 16

8a. CONTRACT OR GRANT NO. N00014-67-A-0112-0011	9a. ORIGINATOR'S REPORT NUMBER(S) 70-16
b. PROJECT NO. NR-047-064	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. AVAILABILITY/LIMITATION NOTICES

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Logistics & Mathematical Statistics Branch Math. Sciences Division. ONR Washington, D. C. 20360

13. ABSTRACT

This paper reviews and compares some new methods which have recently been proposed for updating the inverse in the simplex method. All of these new methods assume an elimination form of inverse, which has lately been shown to be superior to the old product form. Some computational experiments comparing what appears to be the most promising of these new methods with the standard product form method are described and the results appear encouraging. Some suggestions for implementation of this method in a large-scale production LP code are also given. ( )

DD FORM 1 JAN 64 1473

UNCLASSIFIED

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Simplex Method						
Elimination Form of the Inverse (E.F.I.)						
Product Form of the Inverse (P.F.I.)						